File Systems

File Systems — Motivation

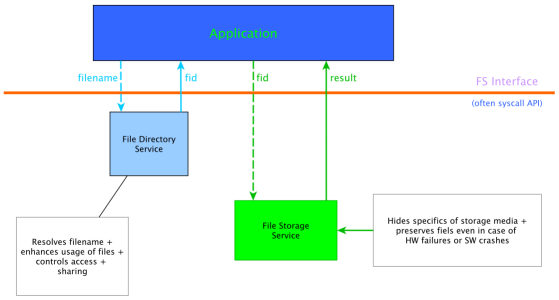
- **Goal**: enable storing of large data amounts
  - store data/program consistently + persistently
  - easily look up previously stored data/program
- **File types**:
  - *data* (numeric, character, binary)
  - *program*

File Systems — Overview

- OS may support multiple file systems
- **Namespace**: all file systems typically bound into single namespace (often hierarchical, rooted tree)

Files – Abstract operations

- **File**: abstract data type/object, offering
  - `create`, `write`, `read`,
  - `reposition` (within file),
  - `delete`, `truncate`,
  - `open`(F_i) (search directory structure on disk for entry F_i, move meta data to memory),
  - `close`(F_i) (move cached meta data of entry F_i in memory to directory structure on disk)



File Management — Goals

- provide convenient file naming scheme
- provide uniform I/O support for variety of storage device types
- provide standardized set of I/O interface functions
- minimize/eliminate loss/corruption of data
- provide I/O support + access control for multiple users
- enhance system administration (e.g., backup)
- provide acceptable performance

File Management — Open files

- several meta data is needed to manage open files
- **File pointer**: pointer to last read/write location, per process that has file opened
- **Access rights**: per-process access mode information
- **File-open count**: counter of number of times a file is opened (to allow removal of data from open-file table when last process closes)
- **Disk location**: cache of data access information

File Access

- **Strictly sequential** (early systems):
  - read all bytes/records from beginning
  - cannot jump round, could only rewind
  - sufficient as long as storage was a tape
- **Random access** (current systems):
  - bytes/records read in any order
  - essential for database systems

Directories — Goals

- **Naming**: convenient to users
  - two users can have same name for different files
  - same file can have several different names
- **Grouping**: logical grouping of files by properties
- **Efficiency**: fast operations

Files — Sharing

- **Issues**:
  - efficiently access to same file?
  - how to determine access rights?
  - management of concurrent accesses?
- **Access rights**:
  - *none*: existence unknown to user, user cannot read directory containing file
  - *knowledge*: user can only determine existence and file ownership
  - *execution*: user can load + execute program, but can not copy it
  - *reading*: user can read file (includes copying + execution)
  - *appending*: user can only add data to file, but cannot modify/delete data in file
  - *updating*: user can modify + delete + add to file (includes creating + removing all data)
  - *change protection*: user can change access rights granted to other users
  - *deletion*: user can delete file
  - *owner*: all previous rights + rights granting
- **Concurrent access**:
  - *application locking*: application can lock entire file or individual records for updating
  - *exclusive vs. shared*: writer lock vs. multiple readers allowed
  - *mandatory vs. advisory*: access denied depending on locks vs. process can decide what to do